STATICALLY ANALYSING THE DYNAMIC BEHAVIOUR OF ASYNCHRONOUS CIRCUITS BY ABSTRACT INTERPRETATION

Sarah Thompson and Alan Mycroft

Computer Laboratory, University of Cambridge, William Gates Building, J J Thomson Avenue, Cambridge

Key words to describe this work: Abstract Interpretation, Asynchronous Circuits, Model Checking, EDA, Static Analysis

Key results: An abstract interpretation framework is described that allows dynamic properties of asynchronous circuits to be statically determined.

How does the work advance the state of the art?: A sound mathematical framework is presented that allows new approaches to the model checking, static analysis and logic synthesis of asynchronous digital circuits.

Motivation (Problems Addressed): Most existing EDA systems are based implicitly on an unsafe assumption that circuits are purely synchronous – the work described here provides a sound basis for the creation of tools that are inherently safe for the general case.

Introduction

Abstract Interpretation is a long-established mathematical approach that has the capability to statically analyse the dynamic properties of a system. Traditionally, it has been applied to software, allowing useful information about the run-time properties of programs to be determined without requiring them to be executed. This paper presents an abstract interpretation framework that allows dynamic properties of asynchronous circuits to be statically determined. We define several alternative models that together form a lattice with Galois connections.

Motivating Example

As a motivating example, the circuit represented by the Boolean expression $a \wedge \neg a$ may at first sight appear to be equivalent to 0 (i.e. ground), due to the Boolean equivalence $a \wedge \neg a = 0$. However, this only holds when a is unchanging – depending on the delays inherent in the circuit, glitches will typically be generated at the output, triggered by either the rising or falling edge of the signal a. If missed at the design stage, such problems must be corrected through simulation and testing, an expensive and time-consuming practice.

Our framework makes it possible to determine what might happen if, for example, a clean transition from 0 to 1 (denoted $\uparrow_0$) is fed into $a \wedge \neg a$:

$$\begin{aligned} \uparrow_0 \wedge \neg \uparrow_0 &= \uparrow_0 \wedge \downarrow_0 && \text{(since } \neg \uparrow_n = \downarrow_n\text{)} \quad (1) \\ &= F_{0..1} && \text{(definition of } \wedge\text{)} \quad (2) \end{aligned}$$

The notation $F_{0..1}$ may be read literally as 'the signal that begins and ends with 0, that contains exactly 0 or 1 complete pulses[1]'. Informally, this may be understood as asserting that the resulting signal *might* be clean – if we

[1]In real circuits, analogue behaviour can result in the generation of extremely brief 'runt pulses'. Our model accommodates this, in that a runt pulse that is below the switching threshold of connected gates is equivalent to F_0, where a runt pulse above the threshold is equivalent to F_1. These alternatives are both captured by the notation $F_{0..1}$.

are lucky with timing – but *can* glitch under the appropriate circumstances. Abstract interpretation allows such potential design errors to be detected by simple calculation, rather than through laborious testing.

The Existing Approach

Current logic synthesis systems typically perform optimisation based on an implicit assumption that generated circuits have a single global clock. Surprisingly, the most commonly used hardware description languages used do not enforce this – in Verilog, for example, it is easy to inadvertently introduce glitches when implementing gated clocks. Even when the source code is correct, optimisation can remove prime implicants that in a synchronous design would be superfluous, but that are essential to the correct functioning of an asynchronous circuit.

Optimising Asynchronous Circuits

When optimizing an asynchronous circuit, it is critically important that any changes made should not introduce new glitches, although removing them is generally safe. It is therefore appropriate to replace $a \wedge \neg a$ with 0, but unwise to replace 0 with $a \wedge \neg a$. Our framework allows such rules to be formally reasoned about. Where a and b are signals, the relation $a \prec b$ asserts that b is 'cleaner' than a, so $F_{0..1} \prec F_0$. Where f and g are functions of signals, $f \prec g$ asserts that g is 'cleaner' that f, for all possible inputs. Since it is possible to prove the correctness of optimisations such as $\forall a \,.\, a \wedge \neg a \preceq F_0$, the technique presented in this paper may also be useful in validating logic synthesis systems that are friendlier to the needs of asynchronous circuit designers. Similar optimisations may also be useful in synchronous design for low power applications – glitches cost power, so their reduction or elimination is likely to result in commensurate improvements in power consumption.

Conclusions

Existing EDA tools typically perform logic synthesis without taking into account asynchronous behaviour, even though this is the default class of circuit described by the two most commonly used hardware description languages, Verilog and VHDL. Our framework provides a sound framework for the construction and validation of new software tools that need not share this limitation.

References

[1] P. Cousot and R. Cousot, *Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints*, Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (Los Angeles, California), ACM Press, New York, NY, 1977, pp. 238–252.

[2] ______, *Systematic design of program analysis frameworks*, Conference Record of the Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (San Antonio, Texas), ACM Press, New York, NY, 1979, pp. 269–282.

[3] A. Mycroft and N. D. Jones, *A relational framework for abstract interpretation*, Lecture Notes in Computer Science: Proc. Copenhagen workshop on programs as data objects, vol. 215, Springer-Verlag, 1984.

[4] S. Thompson and A. Mycroft, *Abstract interpretation of asynchronous circuits*, Designing Correct Circuits (Barcelona), ETAPS, 2004, in preparation.